ISBN 978-987-4417-90-9


# Particle Swarm Optimization with Adaptive Inertia Weight using Fuzzy Logic for Large-Scale Problems

Fabiola-Patricia Paz[a,1], Guillermo Leguizamón[a,2], Efrén Mezura-Montes[b,3]

[a]LIDIC, Universidad Nacional de San Luis, Ejército de Los Andes 950, 5700, San Luis, ARGENTINA - CONICET
[b]Artificial Intelligence Research Center, University of Veracruz, Sebastian Camacho 5, Centro, Xalapa, Veracruz, 91000, MEXICO
[1]fabyppaz@gmail.com.ar, [2]legui@unsl.edu.ar, [3]emezura@uv.mx

**Abstract.** In this paper an alternative approach is proposed to improve the convergence of Particle Swarm Optimization (PSO) algorithm by adapting the inertial weight parameter with a fuzzy logic system to solve large-scale optimization problems. The PSO algorithm is a population-based metaheuristic inspired by the social behavior of birds, and it has been applied to numerous optimization problems successfully. However, one of its main disadvantages is the decaying performance when applied to complex and large-scale problems. The proposed algorithm uses the fuzzy system to dynamically calculate a value of the Inertia Weight parameter during the search process to find better solutions. After carrying out experiments on a well-known benchmark for large-scale optimization, the proposed approach provides a competitive performance.

**Keywords:** Fuzzy logic, Particle Swarm Optimization, convergence control, adaptive inertia weight.

## 1 Introduction

Particle Swarm Optimization (PSO) is an algorithm based on Swarm Intelligence theory, and inspired by the social behavior of certain animals when they interact with another of their same species to achieve a common goal. It was proposed by Kennedy and Eberhart in 1995 and was developed to emulate the cooperative work in bird flocks [1] [2]. Since its original version, PSO has attracted the researchers interest mainly due to its competitive performance in complex search spaces and also its simplicity to code it. However, as other swarm intelligence algorithms, PSO may suffer from premature convergence, especially in complex or large-scale problems [20] [15]. One of the main reasons for this undesired behavior is due to the rapid movement of particles when exchanging information, leading to a low diversity of the swarm in those initial iterations. In recent years, several techniques have been proposed to avoid PSO's premature convergence [3]. In the literature, fuzzy logic has been successfully used to improve the performance of swarm intelligence algorithms. Some published works that were of interest for this work are the following: Olivas et al. [7] [8] [14] implemented PSO and ant colony optimizer (ACO), Pérez et al. [6] the Bat algorithm, Sombra et al. [9] the gravitational search algorithm, Valdez et al. [10] and [11] a set of algorithms such as PSO, genetic algorithm (GA), and ant colony optimization (ACO), Norouzzadeh et al. [12] used PSO, Ochoa et al. [13] adopted differential evolution (DE) and Kumar et al. [16] PSO and other meta-heuristics. However, most of the before mentioned references solved small-scale problems (less than 500 dimensions). On the other hand, large-scale global optimization (LSGO) problems are challenging because the search space grows exponentially as the number of dimensions increases.

Motivated by the above mentioned, in this paper we propose two variants of the PSO algorithm using a fuzzy logic system to solve LSGO problems seeking high-quality solutions with a low computational cost. A set of popular benchmark problems widely used in the literature is solved in this work as they represent different characteristics of real-world instances [17] [20]. Another contribution of the proposal is that two

blurred systems are designed to control the inertia weight with a different approach. The iteration number and the swarm diversity are defined as input variables, while the inertia weight and a variable for resetting a part of the swarm are the output variables. Both, input and output variables are combined into two fuzzy systems designed exclusively to handle LSGO. A comparative study between the systems is then carried out to analyze the quality of the results.

This work is organized as follows: Section 2 presents the Inertia weight PSO algorithm and the Constriction Factor approach. In Section 3, the fuzzy system and its fundamental elements are defined. Section 4 details the parameter control based on fuzzy logic for the PSO. In Section 5 the experiments are conducted. Finally, Section 6 presents the conclusions obtained.

## 2 Particle Swarm Optimization

The PSO algorithm is initialized with a set of randomly generated candidate solutions, called particles, within a D-dimensional search space. Each particle $x_i(t)$ represents a position within the search space, and has its assigned velocity vector $v_i(t)$ and its best position $p_{best_i}(t)$ found so far. The best particle of the whole swarm, based on the fitness function value, is called $\left(g_{best}(t)\right)$. A particle flies to a new position by using Equation (1), called velocity update, and Equation (2), known as flight operator:

$$v_i(t+1) = w * v_i(t) + c_1 * r_1 * \left(p_{best_i}(t) - x_i(t)\right) + c_2 * r_2 * \left(g_{best}(t) - x_i(t)\right) \quad \textbf{(1)}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad \textbf{(2)};$$

where $v_i$ is the velocity of particle $i$, $x_i$ is the current position of particle $i$; $r_1$ and $r_2$ are random numbers with uniform distribution between [0,1]; $p_{best_i}$ is the best position found by particle $i$; and $g_{best_i}$ is the best particle of the swarm. The cognitive and social learning coefficients are represented by $c_1$ and $c_2$, respectively. These values are generally constant, but can also be dynamic. $w$ is a static or dynamic inertia weight [1] [2] [15]. The incorporation of $w$ in the original PSO algorithm allows a better control of the current particle velocities. With this inertia weight, the exploration and exploitation capabilities can be influenced. A higher inertia weight facilitates exploration, while a lower inertia weight favors exploitation. An appropriate $w$-value provides a balance in the search for the best solutions and helps to reduce the computational cost measured by the number of solution evaluations [15].

Another approach used in PSO variants is the constriction factor proposed by Clerc [19]. The velocity equation is defined in Equation (3):

$$v_i(t+1) = X * v_i(t) + c_1 * r_1 * \left(p_{best}(t) - x_i(t)\right) + c_2 * r_2 * \left(g_{best}(t) - x_i(t)\right) \quad \textbf{(3)}$$

and

$$X = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|},$$

where $\varphi = c_1 + c_2$ y $\varphi \geq 4$. The parameter $X$ controls the extent of exploration and exploitation. A $X$ value close to 0 indicates a high degree of exploitation, whereas a value close to 1, a higher degree of exploration is obtained. This value can be constant or can change through the search process, usually with high values in the first iterations and low values late in the search.

## 3 Fuzzy Logic System

Fuzzy Logic (FL) is based on the fuzzy set theory proposed by Lotfi Zadeh in 1965. This approach forms the basis for approximate human reasoning using fuzzy if-then rules as a tool that allows each object to be assigned a degree of membership in one class or another in a smooth and flexible way [4] [5]. Fuzzy inference is the process of obtaining an output value for an input value using fuzzy set theory. There are two types: the Mamdani model and the TSK model (Takagi, Sugeno and Kang). The one proposed by Mamdani in 1975, could be described in four main steps. The first step is the *Fuzzification* of Input Variables, which consists of taking the values of the variables and determining the degree of membership in the fuzzy sets. The next step is the rule *evaluation*. The usual knowledge representation in fuzzy terms is done by **IF A THEN B** rules, where A is the antecedent and B the consequent. If a rule has multiple antecedents, the AND or OR operator is used to obtain a single number that represents the result of the evaluation. This result is applied to the consequent by means of an implication operator. The output *aggregation* consists in the unification of all the output membership functions, combining them to obtain a single fuzzy set. Finally, *defuzzification* is the final result of the system [21].

## 4 Control of Parameters through Fuzzy Logic

A fuzzy system needs to know the condition of the swarm in order to detect swarm stagnation. Inspired by [6] [7] [8] [10] [11] [14], the first input variable is *Iteration*, defined in Equation (**4)** and its value varies between 0 and 1. It can be understood as the progress of the optimization process.

$$Iteration = \frac{iteration_{current}}{Maximum\ of\ iterations} \quad (4)$$

The second input variable is Diversity. Diversity measures the distances between the particles ($x_i$) with respect to the best particle ($G_{best}$) of the swarm at each generation. It is calculated using the Euclidean Distance defined in Equation (**5)**, where *ns* is the number of particles and *nx* is a dimension of particle *i*. The variables $minDiv$ and $maxDiv$ represent the minimum and maximum diversity respectively and are used to normalize diversity at each generation. That is, when $DivNorm$ has a value close to 0 the diversity will be low, otherwise, the diversity will be high [7] [8].

$$Diver = \frac{1}{ns}\sum_{i=1}^{ns}\sqrt{\sum_{d=1}^{nx}\left(x_{i\_d} - G_{best\_d}\right)^2}$$

$$DivNorm = \begin{cases} 0, & \text{when } minDiv = maxDiv \\ \dfrac{Diver - minDiv}{maxDiv - mindiv}, & \text{otherwise} \end{cases} \quad (5)$$

Both, *Iteration* and *Diversity* variables have been used in different approaches [6] [7] [8] [10] [11] [14], and have showed to be suitable indicators for examining swarm conditions during the search process. Therefore, they are adopted in this work.

The objective of the fuzzy system is to control the Inertia Weight to improve particle velocities. Shi et al. [15] use an inertia weight ($w$) that decreases linearly during the iterations and achieved an improved performance in several applications. Eberhart et al. [18] show that $w$ can be configured to be equivalent to the constriction factor equations, then defining the constriction factor as a special case of the inertia weight. Based on these works, an adaptive inertia weight with a constriction factor approach is proposed as an output variable, where $c_1$ and $c_2$ will be constant and $c_1 + c_2 \geq 4$. The velocity would then be defined by Equation (**6)**:

$$v_i(t+1) = \boldsymbol{w} * \left[v_i(t) + c_1 * r_1 * \left(p_{best}(t) - x_i(t)\right) + c_2 * r_2 * \left(g_{best}(t) - x_i(t)\right)\right] \quad (6)$$

the velocity will be limited by the value of $w$ and its value will be defined between [0,1].

Finally, there is a second output variable for the fuzzy system, called *sigma*. This variable is the swarm ratio for the reset of randomly selected particles. This is important because it can potentiate $w$ if, for example, the population starts with a low diversity. Therefore, *Diversity* as well as *Iteration* influence the value that sigma can take. Its maximum proportion value must be chosen carefully. Otherwise the algorithm may never converge. A s*igma* value is empirically defined between [0, 0.2]. Fig. **1** shows the pseudocode of the implementation of the s*igma* variable.

```
particles=permute(numSwarm)
Numpartic=Sigma*numSwarm
 For i=1:Numpartic
  Pos(particles(i))
  Reset Pos
 End
```

**Fig. 1.** Implementation of the s*igma* variable

The membership functions of the fuzzy input and output variables are determined according to previous experiments. In Fig. **2-5** the membership functions for each one of the fuzzy variables are shown.

Both fuzzy systems are of Mamdani type and are ideal for this type of control [6] [7] [8] [10] [11] [14]. The first fuzzy system is called FPSO1 and the second one as FPSO2. Each one of the systems is detailed below.

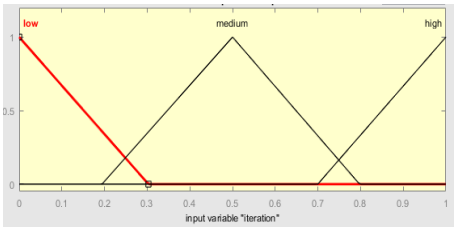


**Fig. 2.** Input variable: *Iteration*

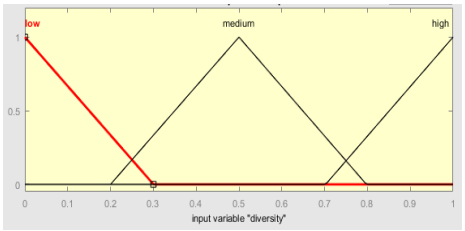


**Fig. 3.** Input variable: *Diversity*

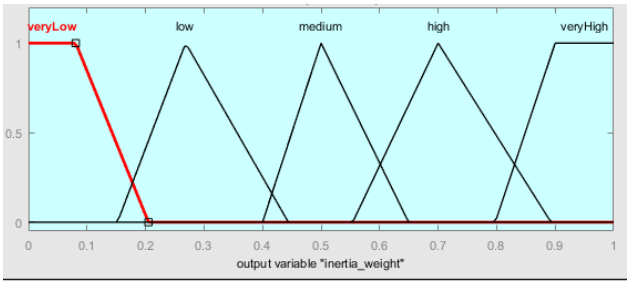


**Fig. 4.** Output variable: *inertia weight*

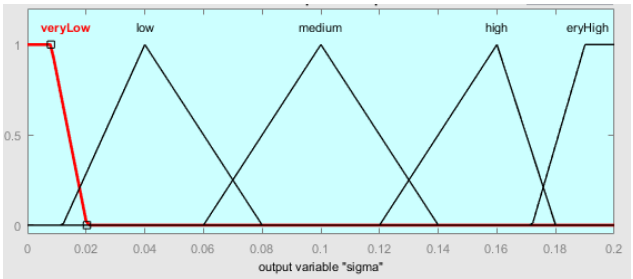


**Fig. 5.** Output variable: *Sigma*

### 3.1 FPSO1

The first variant proposed, called FPSO1, will have two input variables and one output variable, see Fig. **6**. The input variable are *Iteration* and *Diversity* according to Figs. **2** and **3**.

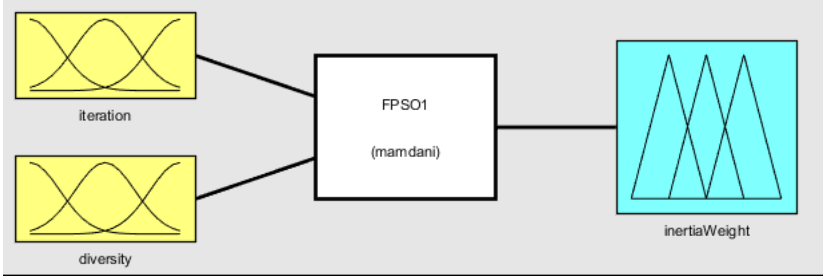


**Fig. 6.** First Fuzzy System FPSO1

The rules of FPSO1 can be seen in Fig. **7**. In general, it is observed that when diversity is low in the initial iterations, the value of the inertia weight must be high so as to help exploring more promising areas. On the other hand, if the diversity is low in the final iterations, the inertia weight must have a very low value to favor exploitation of the area already found. The value assigned for the inertia weight is obtained by the centroid method.

1. If (iteration is low) and (diversity is low) then (inertiaWeight is veryHigh)
2. If (iteration is medium) and (diversity is low) then (inertiaWeight is medium)
3. If (iteration is high) and (diversity is low) then (inertiaWeight is veryLow)
4. If (iteration is low) and (diversity is medium) then (inertiaWeight is high)
5. If (iteration is medium) and (diversity is medium) then (inertiaWeight is medium)
6. If (iteration is high) and (diversity is medium) then (inertiaWeight is veryLow)
7. If (iteration is low) and (diversity is high) then (inertiaWeight is veryHigh)
8. If (iteration is medium) and (diversity is high) then (inertiaWeight is low)
9. If (iteration is high) and (diversity is high) then (inertiaWeight is veryLow)

**Fig. 7.** Rule for fuzzy system FPSO1

### 3.2 FPSO2

The second proposal, called FPSO2, differs from the previous one in the fact that it has an additional output variable. This variable is *Sigma* as can be seen in Fig. **5**. The rest of the variables are the same defined in Fig. **2, 3,** and **4**. Therefore, FPSO2 is defined as shown in Fig. **8** and the rules are shown in Fig. **9**. The value assigned for the output variables (inertia weight and sigma) are obtained by the centroid method.

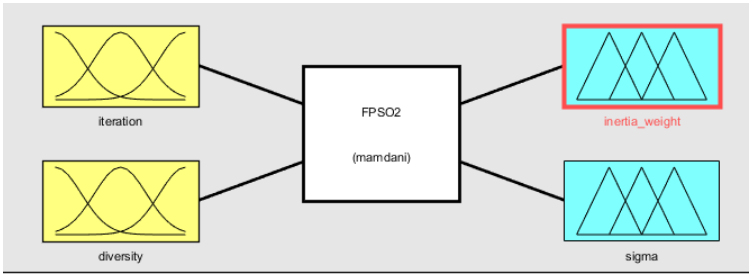


**Fig. 8.** Second fuzzy system FPSO2

1. If (iteration is low) and (diversity is low) then (inertiaWeight is veryHigh) (sigma is veryHigh)
2. If (iteration is medium) and (diversity is low) then (inertiaWeight is medium) (sigma is low)
3. If (iteration is high) and (diversity is low) then (inertiaWeight is veryLow) (sigma is veryLow)
4. If (iteration is low) and (diversity is medium) then (inertiaWeight is high) (sigma is high)
5. If (iteration is medium) and (diversity is medium) then (inertiaWeight is medium) (sigma is medium)

6. If (iteration is high) and (diversity is medium) then (inertiaWeight is veryLow) (sigma is veryLow)
7. If (iteration is low) and (diversity is high) then (inertiaWeight is veryHigh) (sigma is high)
8. If (iteration is medium) and (diversity is high) then (inertiaWeight is medium) (sigma is low)
9. If (iteration is high) and (diversity is high) then (inertiaWeight is veryLow) (sigma is veryLow)

**Fig. 9.** Rule for fuzzy system FPSO2

## 5 PSO variants using FPSO1 and FPSO2

The simple PSO algorithm is combined with the fuzzy systems to generate the following variants:

- *FuzzyPSO1:* This variant is built with the simple PSO algorithm and the fuzzy system FPSO1 and its pseudocode is shown in Fig. **10**.

```
Swarm Initialization
Repeat
  calculate_Iteration Eq. (4)
  calculate_Diversity Eq. (5)
  Fis=FPSO1(Iteration,Diversity)
  Update velocity Eq. (6)
  Update position Eq. (2)
  Update pbest
  Update gbest
Until Maximum_iteration
```

**Fig. 10**. Pseudocode of the *Fuzzy*PSO1 algorithm

- *FuzzyPSO2:* the simple PSO algorithm is combined with the fuzzy system FPSO2 and the pseudocode of this variant can be seen in Fig. **11**.

```
Swarm Initialization
Repeat
  calculate_Iteration Eq.(4)
  calculate_Diversity Eq.(5)
  Fis=FPSO2(Iteration,Diversity)
  Reinitialize particles with sigma
  Update velocity Eq. (6)
  Update position Eq. (2)
  Update pbest
  Update gbest
Until Maximum_iteration
```

**Fig. 11.** Pseudocode of the *Fuzzy*PSO2 algorithm

## 6 Experiments

Table **1** presents seven functions for large-scale global optimization, all of them can scale to different dimensions.

**Table 1.** Large-scale benchmark problems

| | | Function | Limits | Optimum |
|---|---|---|---|---|
| F1 | unimodal | Shifted Sphere Function | $x \in [-100,100]$ | 0 |
| F2 | | Shifted Schwefel's Problem 2.21 | $x \in [-100,100]$ | 0 |
| F3 | multimodal | Shifted Rosenbrock's Function | $x \in [-100,100]$ | 0 |
| F4 | | Shifted Rastrigin's Function | $x \in [-5,5]$ | 0 |
| F5 | | Shifted Griewank's Function | $x \in [-600,600]$ | 0 |
| F6 | | Shifted Ackley's Function | $x \in [-32,32]$ | 0 |
| F7 | | FastFractal "DoubleDip" Function | $x \in [-1,1]$ | unknown |

### 6.1 Experimental Results

The comparisons of the simple PSO algorithm, and the *FuzzyPSO1* and *FuzzyPSO1* proposals were performed with the functions defined in Table **1** for 500 (500 -D) and 1000 (1000-D) dimensions. The following configurations were considered: population size 50 for 500-D and 100 for 1000-D, based on [19] the cognitive learning coefficient $c_1$=2, social learning coefficient $c_2$=3 ($c_1$+$c_2$>4), maximum velocity equal to the maximum value of the variables. There were 5E+03*D fitness evaluations (FEs) and 25 runs for each function as suggested in [20] and the best and mean statistical values were obtained and the better one is highlighted with gray background. The results of the seven test functions in 500-D and 1000-D are shown in Table **2** and representative convergence plots based on the median run of the three algorithms are presented in Fig. **12**.

**Table 2.** Experimental results of F1-F7 in 500-D and 1000-D (mean (best))

| Fun | D | PSO | *FuzzyPSO1* | *FuzzyPSO2* |
|---|---|---|---|---|
| F1 | 500 | 8,930E+05(7,981E+05) | 3,132E+05(2,6E+05) | **2,901E+04 (2,1E+04)** |
| | 1000 | 3,276E+06(2,4E+06) | 1,155E+06(1,0E+06) | **9,227E+04 (7,6E+04)** |
| F2 | 500 | 1,535E+02 (1,5E+02) | 1,350E+02(1,3E+02) | **9,491E+01(9,5E+01)** |
| | 1000 | 1,724E+02 (1,7E+02) | 1,522E+02(1, 5E+02) | **9,535E+01(9,5E+01)** |
| F3 | 500 | 5,318E+11(4,6E+11) | 6,483E+10(3,6E+10) | **1,198E+03(9,0E+02)** |
| | 1000 | 4,188E+12(2,3E+12) | 5,726E+11(4,6E+11) | **3,379E+03(2,7E+03)** |
| F4 | 500 | 7,832E+03(7,4E+03) | 5,338E+03(4,6E+03) | **3,467E+03(2,6E+03)** |
| | 1000 | 1,949E+04(1,7E+04) | 1,427E+04(1,3E+04) | **9,231E+03(8,1E+03)** |
| F5 | 500 | 8,224E+03(7,3E+03) | 2,723E+03(2,2E+03) | **1,989E+02(1,5E+02)** |
| | 1000 | 2,675E+04(2,3E+04) | 1,015E+04(8,7E+03) | **7,828E+02(7,0E+02)** |
| F6 | 500 | 2,080E+01(2,0E+01) | 2,041E+01(2,0E+01) | **1,775E+01(1,7E+01)** |
| | 1000 | 2,144E+01(2,1E+01) | 2,070E+01(2,1E+01) | **1,958E+01(1,9E+01)** |
| F7 | 500 | -3,514E+03(-3,6E+03) | -4,808E+03(-5,0E+03) | **-4,943E+03(-5,6E+03)** |
| | 1000 | -6,646E+03(-6,8E+03) | -8,929E+03(-9,3E+03) | **-9,316E+03(-1,0E+04)** |

The Wilcoxon Signed Rank test [22] was computed to determine if there are significant differences among the proposed algorithms. The results of this test are shown in Table 3 and were calculated using a significance level (**α**) of 0.05. The null hypothesis ($H_0$) indicates that there is no significant difference between the "mean" values of the compared algorithms. While the alternative hypothesis is ($H_1$) indicates that if there are significant differences between the “mean” values of the compared algorithms. Three comparisons were made: (1) *FuzzyPSO1* with *PSO, (2) FuzzyPSO2* with *PSO,* and (3) *FuzzyPSO1* with *FuzzyPSO2.*

**Table 3.** Wilcoxon test results with **α**=0.05

| Algorithms | D | p-value | Decision |
|---|---|---|---|
| FuzzyPSO1 / PSO | 500 | 0.01563 | H0 is rejected |
| | 1000 | 0.01563 | H0 is rejected |
| FuzzyPSO2 / PSO | 500 | 0.01563 | H0 is rejected |
| | 1000 | 0.01563 | H0 is rejected |
| FuzzyPSO2 /FuzzyPSO1 | 500 | 0.01563 | H0 is rejected |
| | 1000 | 0.01563 | H0 is rejected |

*PSO with FuzzyPSO1*: FuzzyPSO1 performed significantly better against PSO. Table **2** shows that the results of FuzzyPSO1 are superior in all functions. However, both algorithms were affected by dimensionality. The graphs in Fig. **12** show that the convergence of FuzzyPSO1 is better and faster than PSO.

*PSO with FuzzyPSO2*: The FuzzyPSO2 variant outperforms simple PSO in the different types of high dimensionality problems. In terms of scalability from 500-D to 1000-D, *FuzzyPSO2* is much less affected than PSO. The convergence of FuzzyPSO2 is better in both unimodal (F1-F2) and multimodal (F3-F7) functions (see Fig. **12**).

*FuzzyPSO1 with FuzzyPSO2*: The performance of FuzzyPSO2 is statistically better than that of *FuzzyPSO1* in 500-D and 1000-D in all 7 test functions. *FuzzyPSO2* converges faster and better on test problems F1, F2, F3, F5, F6. However, in F7 both,

500-D and 1000-D, *FuzzyPSO2* seems to get trapped in local optima in the first cycles, following a convergence to better solutions late in the search. F7 is a complex multimodal function that represents many real-world problems, and both variants had a similar convergence, as seen in Fig.**12**.

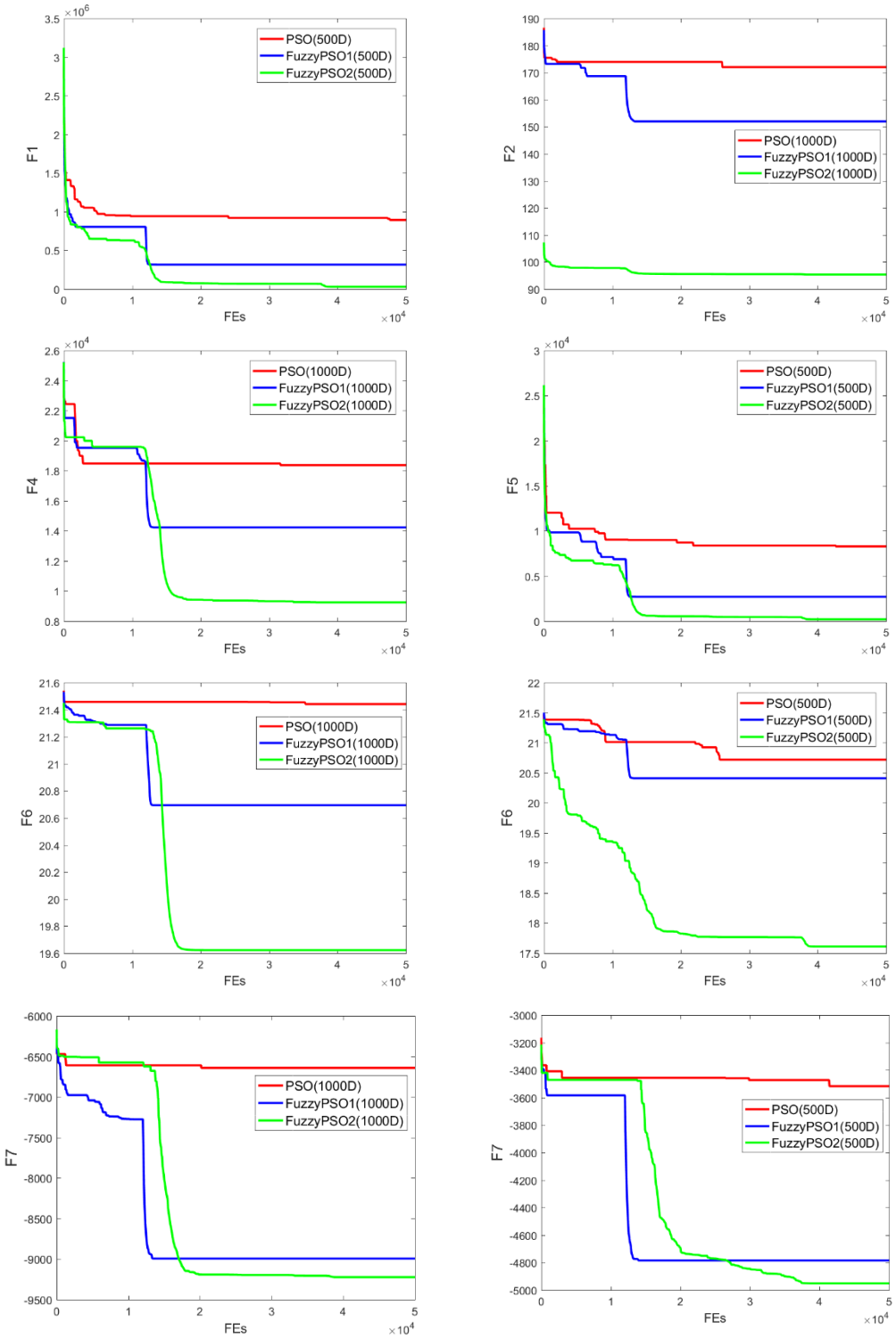


**Fig. 12.** F1-F7 representative convergence plots at 1000-D and 500-D based on median run.

## 7. Conclusions

In this paper two variants named *Fuzzy*PSO1 and *Fuzzy*PSO2 were presented to solve large-scale global optimization problems. The original PSO was combined with the FPSO1 fuzzy system, while the original PSO was merged with the FPSO2 fuzzy system. These systems dynamically manage the inertial weight according to the number of iterations and population diversity. They differ in the fact that FPSO2 restarts a proportion of the population if the diversity is low in the first iterations and that proportion decreases as the iterations increase. It was shown that the *Fuzzy*PSO2 variant had a better performance in all test functions compared to *Fuzzy*PSO1 and a simple PSO. However, in the case of F1 and F3 functions their performance is not yet the expected, and more work is needed to improve the results. Large-scale global optimization problems are challenging and finding the optimal one is not an easy task. However, the results obtained in this initial work are encouraging and suggest that fuzzy concepts can improve the scalability of PSO. As future work we will focus on the performance in unimodal functions and we will also test our approach in large-scale constrained optimization problems.

## References


1. Kennedy, J., Eberhart, R.C. (2001) Swarm Intelligence. Morgan Kaufmann, San Francisco (2001)
2. Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In Proceedings of ICNN'95-International Conference on Neural Networks (Vol. 4, pp. 1942-1948). IEEE.
3. Jie, J., Zeng, J., & Han, C. (2006). Adaptive particle swarm optimization with feedback control of diversity. In International Conference on Intelligent Computing (pp. 81-92). Springer, Berlin, Heidelberg.
4. Zadeh, L.: Fuzzy sets. Information & Control 8, 338–353 (1965). DOI: 10.1016/S0019-9958(65)90241-X.
5. Olivas, J. A. (2001). La lógica borrosa y sus aplicaciones. Pag. Web arantxa. ii. uam. es/dcamacho/lógica/recursos/fuzzy-into-esp. pdf.
6. Pérez, J., Valdez, F., Castillo, O. et al. (2017) Interval type-2 fuzzy logic for dynamic parameter adaptation in the bat algorithm, Soft Computing 21: 667. DOI: 10.1007/s00500-016-2469-3.
7. Olivas, F., Valdez, F., & Castillo, O. (2013). Particle swarm optimization with dynamic parameter adaptation using interval type-2 fuzzy logic for benchmark mathematical functions. In 2013 World Congress on Nature and Biologically Inspired Computing (pp. 36-40). IEEE.
8. Olivas, F., & Castillo, O. (2013). Particle swarm optimization with dynamic parameter adaptation using fuzzy logic for benchmark mathematical functions. In Recent Advances on Hybrid Intelligent Systems (pp. 247-258). Springer, Berlin, Heidelberg.
9. Sombra, A., Valdez, F., Melín, P., & Castillo, O. (2013). A new gravitational search algorithm using fuzzy logic to parameter adaptation. In 2013 IEEE Congress on Evolutionary Computation (pp. 1068-1074). IEEE.
10. Valdez, F., Melín, P., & Castillo, O. (2011). An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms. Applied Soft Computing, 11(2), 2625-2632.
11. Valdez, F., Melín, P., & Castillo, O. (2014). A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation. Expert systems with applications, 41(14), 6459-6466.
12. Norouzzadeh, M. S., Ahmadzadeh, M. R., & Palhang, M. (2012). LADPSO: using fuzzy logic to conduct PSO algorithm. Applied Intelligence, 37(2), 290-304.
13. Ochoa, P., Castillo, O., & Soria, J. (2017). Differential evolution using fuzzy logic and a comparative study with other metaheuristics. In Nature-Inspired Design of Hybrid Intelligent Systems (pp. 257-268). Springer, Cham.
14. Olivas, F., Valdez, F., Castillo, O., Gonzalez, C. I., Martínez, G., & Melín, P. (2017). Ant colony optimization with dynamic parameter adaptation based on interval type-2 fuzzy logic systems. Applied Soft Computing, 53, 74-87.
15. Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In International conference on evolutionary programming (pp. 591-600). Springer, Berlin, Heidelberg.

16. Kumar, S., & Chaturvedi, D. K. (2011). Tuning of particle swarm optimization parameter using fuzzy logic. In 2011 International Conference on Communication Systems and Network Technologies (pp. 174-179). IEEE.
17. Awad, N. H., Ali, M. Z., Liang, J. J., Qu, B. Y., Suganthan, P. N., & Definitions, P. (2016). Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Nanyang Technological University, Jordan University of Science and Technology and Zhengzhou University. Tech. Rep.
18. Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512) (Vol. 1, pp. 84-88). IEEE.
19. Clerc, M. (1999) The swarm and the queen: towards a deterministic and adaptive particle swarm optimization, Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), Washington, DC, USA, 1999, pp. 1951-1957 Vol. 3, doi: 10.1109/CEC.1999.785513.
20. Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., & Yang, Z. (2007). Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Nature Inspired Computation and Applications Laboratory, USTC, China, 24.
21. González Morcillo C. (2005). Lógica Difusa. Una introducción práctica. Técnicas de Soft Computing.
22. Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation, 1(1), 3-18.
23. Li, X., & Yao, X. (2011). Cooperatively coevolving particle swarms for large scale optimization. IEEE Transactions on Evolutionary Computation, 16(2), 210-224.